A Self-organizing Approach for Building and Maintaining Knowledge Networks

Gabriella Castelli, Marco Mamei, and Franco Zambonelli

DISMI - Dipartimento di Scienze e Metodi dell'Ingegneria
University of Modena and Reggio Emilia
Via Amendola 2 , 42100 Reggio Emilia, Italy
{gabriella.castelli,marco.mamei,franco.zambonelli}@unimore.it

Abstract. Pervasive and mobile devices can generate huge amounts of contextual data, from which knowledge about situations occurring in the world can be inferred for the use of pervasive services. Due to the overwhelming amount of data and the distributed and dynamic nature of pervasive systems, this may be not a trivial task. Indeed the management of contextual data should be run by a dedicate middleware layer, i.e., knowledge networks in charge of organizing and aggregating such data to facilitate its exploitation by pervasive services. In this paper we introduce a unsupervised, distributed and self-organizing approach to build and maintain such a layer based on simple agents that organize and extract useful information from the data space. We also present a Java-based implementation of the approach and discuss experimental results.

Keywords: Distributed Middleware, Knowledge Networks, Context Awareness.

1 Introduction

Pervasive and mobile devices and Web 2.0 are already able to generate an overwhelming amount of data about users and context, from which knowledge about situations and facts occurring in the world should be inferred for the use of pervasive and mobile services. A service in need of understanding what is happening around can access the produced pieces of data and analyze them to realize what is the current situation of its context. Nevertheless there are a number of complexities inherent in this process, such as the communication efforts to retrieve the useful knowledge out of an overwhelming amount of data, and the computational efforts to analyze, relate and aggregate such information.

Accordingly, a real challenge for future pervasive services is the investigation of principles, algorithms, and middleware infrastructures, via which this growing amount of distributed information can be properly represented, organized, aggregated, and made more meaningful, so as to facilitate the successful retrieval by pervasive services [7,3]. Many approaches [10,4] are currently going in the direction of adopting specific middleware layer, i.e., a knowledge network layer embedding data and algorithms, and providing effective access to such data by services.

Y. Cai et al. (Eds.): Mobilware 2010, LNICST 48, pp. 175–188, 2010.

Unlike other approaches (e.g., centralized and/or deterministic) to data organization and aggregation [24,5], we propose a distributed self-organized approach to organize, link, and aggregate, related items of contextual information. This choice better suits the decentralized, dynamic, and unpredictable nature of pervasive and mobile systems. In particular, in this paper:

1. We present a middleware architecture and prototype to store and manage contextual data coming from diverse pervasive devices structured according to the W4 data model [9]
2. We present an original self-organized algorithmic approach to perform knowledge networking over a massive amount of distributed pieces of knowledge stored in the above middleware
3. We show a several experiments we performed to test the system and discuss evaluation results.

The remainder of this paper is organized as follows. Section 2 briefly summarizes the W4 data model that is used to represent contextual data provided by pervasive devices and the middleware architecture. In Section 3 introduce the W4 Knowledge Networks idea, describe the algorithmic approach to organize isolated and distributed pieces of paper into networks of correlated data items. Section 4 presents the performance evaluation. Section 5 discusses related work, and finally Section 6 concludes.

2 The W4 Model and Architecture

We adopt the W4 data model to represent and structure the data to to illustrate our self-organizing approach for building and maintaining knowledge networks. The W4 data model has been firstly described in [9]. Here we shortly summarize its key features and then give an overall architectural view of the W4 middleware, its API, ant its implementation.

2.1 The W4 Data Model and Architecture

The proposed W4 model starts from the consideration that any elementary data atoms as well as any higher-level piece of contextual knowledge, in the end, represents a fact which has occurred in the world. Such facts can be expressed by means of a simple yet expressive four-fields tuples (Who, What, Where, When): *"someone or something (Who) does/did some activity (What) in a certain place (Where) at a specific time (When)"*.

More in particular the four-fields of the W4 data model each describes a different aspect of a contextual fact:

- The *Who* field associates a subject to a fact. The *Who* field is represented by a type-value pair.
- The *What* field describes the activity performed by the subject. It is represented as a string containing a predicate:complement statement.

- The *Where* field associates a location to the fact. In our model the location may be a physical point represented by its coordinates, a geographic region, or it can also be a place label.
- The *When* field associates a time or a time range to a fact. This may be an exact time/time range or a context-dependent expression, e.g., *now*.

The way it structures and organizes information makes the W4 data model able to represent data coming from very heterogeneous sources and simple enough to promote ease of management and processing (although we are perfectly aware that it cannot capture each and every aspect of context, as freshness of data, reliability, access control, etc).

2.2 The W4 API

In the W4 data model, we rely on the reasonable assumption that software drivers are associated with data sources and are in charge of creating W4 tuples and inserting them in some sorts of shared data spaces that are distributed in physical world.

The interface to access the W4 middleware took inspiration from tuple-space approaches [1] and consists in two basic operation:

```
void inject(KnowledgeAtom a);
KnowledgeAtom[] read(KnowledgeAtom a);
```

The *inject* operation is equivalent to a tuple space "out" operation: an agent accesses the closest data space to store a W4 tuple there.

The *read* operation is used to retrieve tuples from the closest data space via querying. A query is represented in its turn as a W4 template tuple. Upon invocation, the read operation triggers a pattern-matching procedure between the template and the W4 tuples that already populate the data space. Pattern-matching operations work rather differently from the traditional tuple space model and may exploit differentiated mechanisms for the various W4 fields. Read operations can involve searching in multiple tuple spaces, as explained in Section 3.3.

In [7] we provide several examples of knowledge representation and knowledge generation using the W4 data model.

2.3 Architecture and Implementation

Figure 1 depicts the overall architecture of a W4 system.

At the bottom there are diverse data sources that produce data formatted according the W4 data model and feed a number of W4 tuple spaces. We assume that software drivers gather information from all the available devices (e.g., RFID tags, GPS devices, Web services), and combine them with the goal of producing a W4 tuple as accurate and complete as possible.

The W4 system is made up by a number of distributed W4 tuple spaces. Those tuple spaces can be both local tuple spaces hosted by personal devices

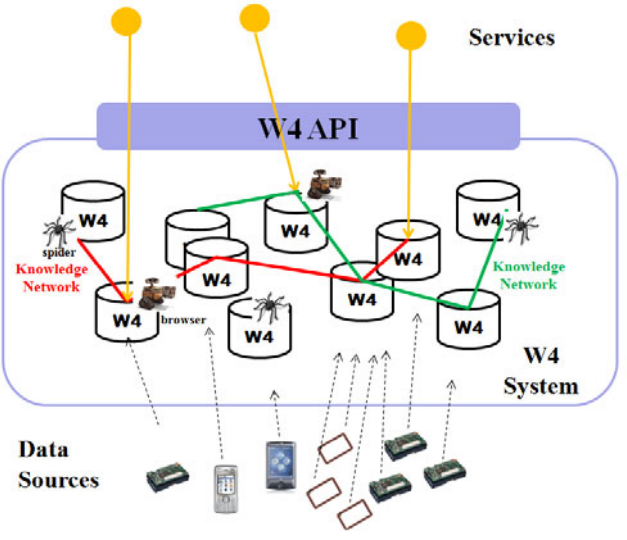

Fig. 1. The W4 System Architecture

and shared tuple spaces acting as public accessible servers. In the systems there are a variety of agents that access the tuple spaces via the W4 API in order to organize the data layer. In particular:

- *Spiders*: are able to jump from a tuple space to another and link tuples that are related into knowledge networks
- *Browsers*: can browse a knowledge network to solve a query and to infer new tuples

Many W4 Knowledge Networks can be realized and coexist in the W4 system, each realizing a specific view over the data. Those agents (i.e., spiders and browsers) and the algorithms to create and manage knowledge networks will be presented in Section 3.

Finally, at the top there are the various services that access the W4 system to retrieve data, to whom the internal W4 system and data location are completely transparent. Indeed they can act over the system submitting queries to the closest W4 tuple space via the W4 API.

We developed a prototype implementation of the described architecture in a small pervasive computing testbed by extending the LighTS Tuple Space [2], a light weight tuple space implementation particularly suitable for context-aware application , and by realizing spiders and browsers as simple Java agents.

The implemented W4 middleware runs on laptops and on PDAs equipped with wireless interface and J2ME-CDC (Personal Profile) Java virtual machine.

3 W4 Knowledge Networks

Although the W4 data model proved to be rather flexible to manage contextual data, the idea is to exploit the W4 structure to access and exploit distributed contextual data in a more sophisticated and effective way. More specifically we propose general-purpose mechanisms and policies to link together knowledge atoms, and thus form W4 Knowledge Networks in which it will be possible to navigate from a W4 tuple to the others. Moreover, new information could be produced combining and aggregating existing tuples while navigating the space of W4 tuples.

The basic ideas towards the realization of W4 knowledge networks had been anticipated also in [9] and [7], but only in this paper they are eventually realized and evaluated.

3.1 The W4 Knowledge Networks Idea

The W4 Knowledge Networks approach is based on the consideration that a relationship between knowledge atoms can be detected by a relationship (i.e., a pattern-matching) between the information contained in the atoms fields. In particular, for the W4 data model, we can identify two types of pattern matching relations between knowledge atoms:

- *Same value – same field*: We can link together those W4 tuples in which the values in the same field match according to some pattern-matching function. In this way, we can render complex concepts related to groups of W4 tuples, e.g. *All students (same subject) who are attending a class (same activity) at the same room (same location).*
- *Same value – different field*: We can link atoms in which the same information appears in different fields augmenting the expressive level of the information contained in the W4 tuples. For example, a knowledge atom having *When: 18/09/2009* can be linked with another atom like *Who: Fall Class Begin* , to add semantic information to that date.

Exploiting those correlations it is possible to find the relationships between one particular W4 tuple with other tuples in the data space, which may then be used to create a web of linked information both to more effectively navigate in the space of information (e.g., for effectively gathering information correlated to a specific context) and as a basis for more elaborated inference and reasoning (e.g., for representing in a comprehensive and expressive way complex situations).

3.2 The W4 Knowledge Networks Algorithm

An unsupervised, distributed and self-organizing approach to generate and maintain the knowledge networks' layer is clearly required by the decentralized nature of pervasive computing systems and the overwhelming amount of generated data, which prevent the use of a centralized process for data management. To this end, we adopt a swarm-based approach relying on a two-phase process.

The first phase is the identification of all possible correlations of interest between knowledge atoms, and the creation of links between W4 atoms. This can be done by a number of simple agents, which we call *spiders* as they weave their webs between correlated tuples. Each spider is associated with a pattern matching function that takes a W4 tuple and matches it against a W4 tuple used as a template. The function returns a boolean value meaning wether the matching is successful or not, and accordingly suggesting creating a link or not. Obviously the simpler the matching function is (i.e., few fields of a W4 tuple are involved), the more the resulting net of links can be reusable. E.g., A1 is in charge of linking together all the tuples with corresponding *who* fields, for instance all the tuples whose who field corresponds to *user:Gabriella*, while another spider agent can search tuples with corresponding where field.

Spiders continuously surf W4 tuple spaces in order to retrieve tuples that fulfill the specific relationship, those tuples are virtually linked together thus creating a W4 knowledge network for the given relationship. To this end spiders must be capable of analyzing W4 tuples stored in different tuple spaces and building correlation networks that extends over distributed tuple spaces. For this reason, spiders are realized in terms of weakly mobile java agents [13].

The spiders' algorithm follows:

```
define:
    rel; //the relation to be satisfied
    knet; //the knowledge network reference

Main:
    Do forever:
        TupleSpace ts = random();
        move (ts);
        tuple t[] = ts.read(rel);
        knet.add(t);

    Done;
```

The spider chooses a random tuple space and checks if any tuple in the tuple space fulfills the given relationship. If it is positive the tuples are added to the knowledge network *knet* by adding a reference to the last tuple space that was found earlier, i.e., drawing a link between the last tuple space added to the knowledge network *rel* and the current one. This process continuously repeats. In this way, a single knowledge network of links between correlated tuples is generated. More spiders can work concurrently both on the same relationship or on different ones, building the knowledge networks layer in a self-organizing fashion.

The second step is the generation of new knowledge atoms, by analyzing which of the identified links can lead to a new W4 atom as a process of merging related atoms. This activity is performed by another class of agents, called *browsers*. Browser agent surf the knowledge networks trying to generate new W4 atoms. Each browser is capable of inferencing a specific type of relationship. The browsers' algorithm follows:

```
define:
    rel; // the relation that the browser is capable to infer

Main:
    Do forever:
        TupleSpace ts = random();
        tuple t = ts.random();
        ts.add (GenerateNewKnowledge(t));
    Done;
```

The browser chooses a random tuple t in the system, and locates all the knowledge networks in which the tuple t is involved. Then the browser start to browse each of the found knowledge networks. For each tuple ti found in a related knowledge networks, the browser checks if he is able of generate a new W4 atom carrying higher knowledge. If positive, the new atom is generated and added to the current tuple space. The issue of bounding the amount of knowledge generation has been discussed in [8].

3.3 Using W4 Knowledge Networks

The idea at the base of the W4 Knowledge Networks approach is that spiders and browsers continuously surf, analyze, correlate and infer new knowledge. In this way new tuples are linked to the knowledge networks of interest and new knowledge networks can be realized as soon as they become of interest for services that access the data middleware. At the same time, browsers can exploit the knowledge networks to browse among tuples that are somehow related and possibly infer new knowledge to be injected in the system in form of a W4 tuple.

Although the knowledge networks can be used as the basis for knowledge reasoning, even when new data are not generated, the web of links between atoms can be fruitfully used during querying to access and retrieve contextual information more effectually. When a query is submitted to the W4 tuple space system, a query-solving agent capable of browsing knowledge networks, i.e. a query solving browser created in order to solve a query, analyze the query template and determine one or more knowledge networks to which the matching tuples should belong. Then the query solving browser choose a random W4 tuple space in the system and scans it until he finds an entry point for one of the identified knowledge networks, i.e. a tuple belonging to one of those knowledge networks. When the entry point is found, the agent starts to jump from the entry point tuple to the other tuples in the identified knowledge network, checking if they matches the template and finally returns the retrieved tuples. This is beneficial for services because fewer read operations have to be performed when exploiting knowledge networks instead of a set of data spaces in which information is not related to each other.

4 Performance Evaluation

To assess the W4 approach presentation, we report several experimental results we performed to evaluate the effectiveness and feasibility of the proposed approach and compare it to other solutions.

To test the approach, we developed a simulated environment based on the Repast framework [http://repast.sourceforge.net] and integrated with the actual prototype presented in Section 2.3. The simulated environment is used to generate a huge amount of data that are needed in order to properly test our middleware. We represented a virtual campus with a number of users (i.e., professors, students, administrative staff, etc.) each moving in the environment and performing their day by day activities. The virtual campus is split in 100 zones, each of them holds a private W4 tuple space that stores all the tuples generated in it. Periodically a W4 tuple for each user is generated on the basis of the current position, activity and time. In this scenario many tuples are stored in the W4 tuple spaces, and services may find difficult to access those data.

4.1 Efficiency

The first set of experiments aims to measure the efficiency of the W4 system in retrieving information and comparing it with an exhaustive search in tuple spaces and with an hash based approaches based on the performance of the above systems when a non destructive query is submitted to the system.

The exhaustive search is performed on a tuple space that embeds the W4 data model facilities but not the W4 knowledge networks mechanisms. When a query is submitted to this simplified W4 tuple space system, a query agent chose a random tuple space in the system and scans it seeking for the W4 tuples that fulfill the query template. Then a random tuple space is chosen again, until the whole system is scanned.

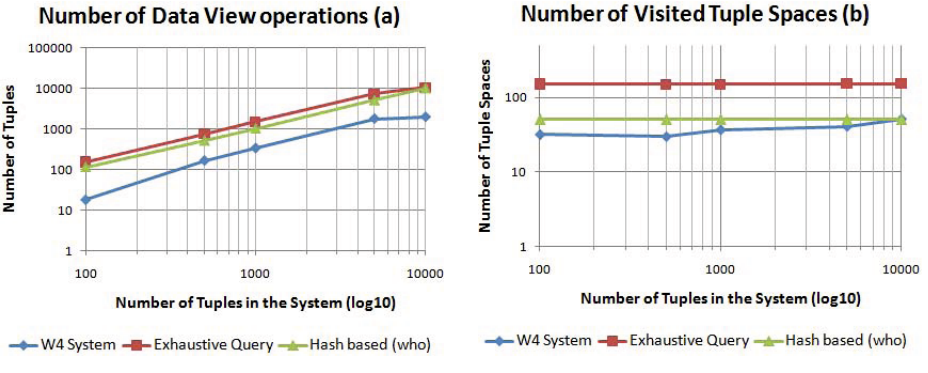

Fig. 2. Efficiency: (a) Number of view operations done by query-solving agents. (b) Number of view operations done by query agents.

Hash based tuple space is a well known and popular technique for data indexing in distributed environment. Here we follow an approach similar to [15] in which a single field of the tuple structure is used for the hashing operation and indexing purpose. When a tuple is injected in the distributed system, the hashing operation is performed over the designated field and the tuple is then stored on the resulting tuple space. Then when a query is submitted to the system, the same hashing operation is performed on the query template and the result indicates the tuple space to scan for results. In this simulations we considered the hash performed on the *who* field. Similar results are achieved considering the others fields of a W4 tuple. Of course other hash based approaches that use tuple spaces exist. We intentionally didn't consider approaches that consider to hash more than one W field at a time, because the issues that would arise in their distributed management make them ineffective in highly dynamic scenarios.

The experiment works as follow: we fed the W4 system with fixed amount of tuples and made them organized in W4 knowledge networks. Then, for the sake of experiments, we stopped the data sources and submitted to the system the following complex query: "Retrieve all the users that were near agent A5 was, on time timeT1". For a W4 System this means that the following two queries should be subsequently resolved:

```
query1 (who, what, where, when) = (user:A5, *, ?var1, timeT1)
```

The second query looks as follow:

```
query2 (who, what, where, when) = (*, *, var1, timeT1)
```

Here both the *where* is automatically considered as a bounding box and the *when* field is automatically transformed in a time interval. To solve such a query two knowledge networks must been investigated: one relating all tuples from *user:A5*, and one relating all tuples that refers a specific region of the space.

We run the simulations 15 times and depicted the average values. Figure 2 (a) shows the number of tuple spaces visited by the query-solving agent in the considered systems. The W4 tuple space systems performs better than the other considered approaches. Indeed, in the medium case, the exhaustive search has to query half the number of tuple spaces in the system to solve the first sub-query and the whole number of the tuple spaces to solve the second ones. The hash approach works better than the exhaustive query because one of the sub-query is solved thanks to the hashing operation, nevertheless the other sub-query have to be solved traditional as in the case of the exhaustive search. However exploiting the W4 Knowledge Networks is even better because the number of accessed tuple spaces is determined by the number of tuple spaces involved in the knowledge networks of interest.

Figure 2 (b) shows the number of read operation performed. Also in this case the W4 tuple space system performs better then the other systems. As in the previous case, the exhaustive search have to access half the number of tuples in the systems to solve the first sub-query, and all the tuples in the system to solve the second one. Here the performance of the hash based systems can be significantly different depending if the hashing is performed on the who field

or on the where field. However, when data are accessed on multiple semantic dimensions, the W4 system performs better because all the fields are considered equally important when building knowledge networks.

4.2 Effectiveness

Provided that the W4 approach exhibits a good behavior in accessing contextual data (i.e., the access costs are lower than the other considered approaches), we run a second set of experiments to test the effectiveness of the knowledge networks' approach, in terms of accuracy of provided results when the knowledge networks algorithms are running, i.e. the fraction of the documents that are relevant to the query that are successfully retrieved (also called "recall" in information retrieval).

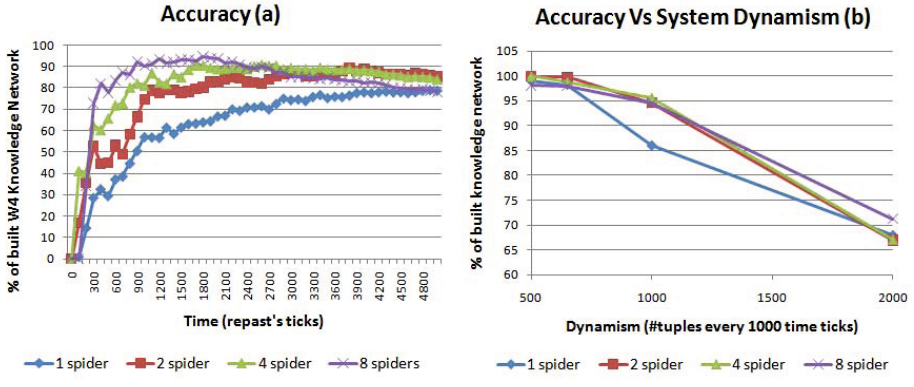

Fig. 3. (a) Accuracy of the indexation over time. (b) Accuracy of the indexation Vs dynamism of the system.

In order to test the effectiveness of the approach, we started feeding the system with W4 tuples generated by the simulated environment and let the knowledge networks keep organizing. Periodically we checked the content of a specific knowledge networks respect to the content of the whole W4 system to measure the percentage of tuples that has been indexed.

Of course the indexing works as quicker as more spiders are involved in, to this end Figure 3 (a) compares the results obtained when a different number of spiders is running. The obvious result is that the more spiders are working, the quicker the knowledge network reaches its indexation level. We can see that it takes a certain amount of time for the knowledge network to reach its stable value of indexation that is in the satisfactory range of 80-90% depending on the number of spiders run. This suggests that the W4 system could be improved by taking into account the W4 tuples' injection rate in order to autonomously determine the right number of spiders running.

Accordingly to this observation, we performed a second set of experiments varying the dynamism (i.e. the tuples injection rate) of the system respect to

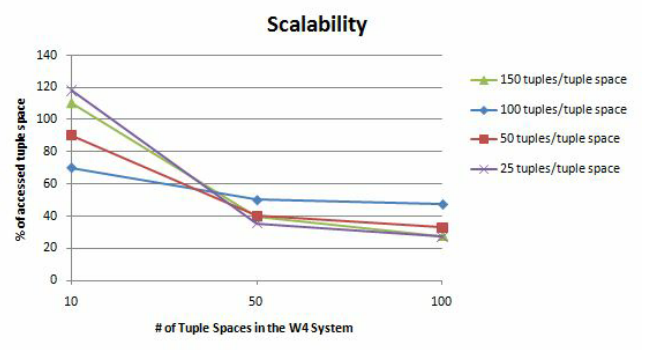

Fig. 4. Scalability of the w4 systems respect to the number of tuple spaces in the system (number of tuples per tuple space fixed)

the number of spiders running, the results is quite interesting because they give an idea of the number of spiders that should run simultaneously in relation to the dynamism of the system. As we expected, Figure 3 (b) shows that as the dynamism of the system increases, i.e. tuples are injected in the W4 system more quickly, the percentage of indexation decrease. That is, when the tuple injection rate increase, it may be needed to run more spiders and browsers to keep the good level of indexation.

4.3 Scalability

Another key factor in distributed systems is their ability to scale. To test further the system scalability we performed another set of experiments fixing the number of tuples per tuple space, and varying the number of tuple space in the W4 system. We performed the measurements as described for the efficiency experiments, and measured the percentage of tuple spaces accessed to solve the query (the number of tuples accessed is not represented because it is highly correlated, as shown in Figure 2). Figure 4 depicts results. We can see that the performances improve when the number of tuple spaces in the system increase. This is due to the fact that the more the system is wide and distributed, the more selective the knowledge networks can be. Indeed knowledge networks approach can be useful only if accessing the knowledge networks allows to skip out accessing the majority of tuple spaces (and then tuples) in the systems, i.e., the knowledge networks extends over a limited subset of tuple spaces of the whole system. In other words the W4 approach makes sense in a distributed environment rather than in a centralized and static one.

5 Related Work

Context is a very fluid notion and although several researchers claim that it is very hard to abstract it in terms of variables and data models [12], it is also a

widespread opinion that a more pragmatic perspective should be adopted. Early works in this area, as from Schmidt et al. [23] and Dey et al. [11], concentrates on the issue of acquiring context data from sensors and of processing such data but they generally miss in identifying a uniform model to describe the data and analyzing the issues at the middleware level. Some recent proposals, such as [24,5] focus on providing models for contextual data that adopt a uniform well-defined structure. Indeed, our W4 proposal accounts for a very similar structuring for contextual information, and enriches it further with a well-defined API, and with the possibility of linking data atoms and of providing application-specific views to services.

An increasing number of research works get inspiration from tuple space middleware models [1] and propose representing and storing contextual information in the form of tuples to be stored in distributed tuple spaces. Egospaces [16] adopts this perspective, without committing to a specific pre-defined structure for context tuples, which can make it difficult for services to uniformly deal with tuples represented in different formats. Other proposal, such as The Context Fabric model [14] rely on well-structured context tuples. Recent proposals focusing on sensor networks, suggest exploiting a tuple-based approach to provide application-specific views on sensorial data [19]. In general we consider tuple-based approaches very suitable for organizing and accessing contextual information, but we also think that there is need of more structuring and flexibility than those exhibited by the existing approaches.

In the above described work, the issue of relating contextual data atoms with each other and of providing different views to different applications is not generally addressed. More recently, other proposals have adopted a similar endeavor but have considered the issue of adopting specific ontologies to model context information and enable other than efficient querying also efficient context-reasoning [22,17]. Although such approaches tend to be too application-specific, they attribute the importance of linking independent atoms of contextual information (with ontological relations) and of reasoning not only on individual data items but also on their relations, an idea which is fully shared by our knowledge network vision. Other proposals experience different techniques for context reasoning. Many works, such as and [21], are focused on situation learning and situation relationships in smart environment. Other works, such as [20] propose predicate logic as an effective language for context-aware reasoning. The W4 Knowledge Networks approach we propose aims to be more general and proposes an approach different from traditional ones, considering self-organizing agents. Campbell et al. [6] consider the possibility of extracting higher-level knowledge from raw sensed data merging feature vectors in an opportunistic fashion for people-centric application. The idea of merging and considering data coming from diverse sources is shared with the W4 Knowledge Networks approach. However in the W4 approach we go further considering multiple knowledge views that can be accessed by multiple services.

Obviously also other areas of research contributed towards the realization of our knowledge networks vision, in particular data mining and pattern discovery and granular computing. See [7] for a critical survey.

6 Conclusion and Future Works

Despite the promising results achieved so far in the study of the W4 self-organized knowledge networks algorithms, some research issues still have to be faced. In particular, more experiments should be done to evaluate properly the overhead coasts.

Moreover, in the current implementation of the W4 system, the number of tuples stored in the system is constantly increasing as new data are injected in the system. There is the need for a "garbage collection" solution and we plan to experiment with a concept of knowledge tuple fading as introduced in [18]. Finally, security and privacy issues need to be analyzed w.r.t. accessing W4 tuples and their relations.

References

1. Ahuja, S., Carriero, N., Gelernter, D.: Linda and friends. Computer 19(8-9), 26–34 (1986)
2. Balzarotti, D., Costa, P., Picco, G.P.: The LighTS Tuple Space Framework and its Customization for Context-Aware Applications. International Journal on Web Intelligence and Agent Systems 50(1-2), 36–50 (2007)
3. Bettini, C., Brdiczka, O., Henricksenc, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing (in press)
4. Bicocchi, N., Castelli, G., Mamei, M., Rosi, A., Zambonelli, F., Baumgarten, M., Mulvenna, M.: Knowledge networks for pervasive services. In: Proceedings of the 2009 International Conference on Pervasive Services, ICPS 2009, pp. 103–112. ACM, New York (2009)
5. Bravo, J., Hervs, R., Snchez, I., Chavira, G., Nava, S.: Visualization services in a conference context: An approach by rfid technology. Journal of Universal Computer Science 12(3), 270–283 (2006)
6. Campbell, A., Eisenman, S., Lane, N., Miluzzo, E., Peterson, R., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G.-S.: The rise of people-centric sensing. IEEE Internet Computing 12(4), 12–21 (2008)
7. Castelli, G., Mamei, M., Zambonelli, F.: Engineering contextual knowledge for autonomic pervasive services. International Journal of Information and Software Technology 52(8-9), 443–460 (2008)
8. Castelli, G., Menezes, R., Zambonelli, F.: Self-organized control of knowledge generation in pervasive computing systems. In: ACM Symposium on Applied Computing, March 8-12 (2009)
9. Castelli, G., Rosi, A., Mamei, M., Zambonelli, F.: A simple model and infrastructure for context-aware browsing of the world. In: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications, PERCOM 2007, Washington, DC, USA, pp. 229–238. IEEE Computer Society Press, Los Alamitos (2007)

10. Clark, D.D., Partridge, C., Ramming, J.C., Wroclawski, J.T.: A knowledge plane for the internet. In: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM 2003, pp. 3–10. ACM, New York (2003)
11. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human Computer Interaction 16(2), 97–166 (2001)
12. Dourish, P.: What we talk about when we talk about context. Personal Ubiquitous Computing 8(1), 19–30 (2004)
13. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Transactions on Software Engineering 24, 342–361 (1998)
14. Hong, J.I.: The context fabric: an infrastructure for context-aware computing. In: extended abstracts on Human factors in computing systems, CHI 2002, pp. 554–555 (2002)
15. Jiang, Y., Xue, G., Jia, Z., You, J.: Dtuples: A distributed hash table based tuple space service for distributed coordination. In: Grid and Cooperative Computing, 2006, pp. 101–106 (October 2006)
16. Julien, C., Roman, G.-C.: Egospaces: facilitating rapid development of context-aware mobile applications. IEEE Transactions on Software Engineering 32(5), 281–298 (2006)
17. Lee, D., Meier, R.: Primary-context model and ontology: A combined approach for pervasive transportation services. In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2007. PerCom Workshops 2007, pp. 419–424 (2007)
18. Menezes, R., Wood, A.: The fading concept in tuple-space systems. In: Proceedings of the 2006 ACM Symposium on Applied Computing, Dijon, France, pp. 440–444. ACM Press, New York (2006)
19. Mottola, L., Picco, G.P.: Logical neighborhoods: A programming abstraction for wireless sensor networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 150–168. Springer, Heidelberg (2006)
20. Ranganathan, A., Campbell, R.H.: An infrastructure for context-awareness based on first order logic. Personal Ubiquitous Comput. 7(6), 353–364 (2003)
21. Reignier, P., Brdiczka, O., Vaufreydaz, D., Crowley, J.L., Maisonnasse, J.: Context-aware environments: from specification to implementation. Expert Systems: The Journal of Knowledge Engineering 24(5), 305–320 (2007)
22. Roussaki, I., Strimpakou, M., Kalatzis, N., Anagnostou, M., Pils, C.: Hybrid context modeling: A location-based scheme using ontologies. In: IEEE International Conference on Pervasive Computing and Communications Workshops, vol. 1, pp. 2–7 (2006)
23. Schmidt, A., Aidoo, K.A., Takaluoma, A., Tuomela, U., Laerhoven, K.V., de Velde, W.V.: Advanced interaction in context. In: Gellersen, H.-W. (ed.) HUC 1999. LNCS, vol. 1707, p. 89. Springer, Heidelberg (1999)
24. Xu, C., Cheung, S.C.: Inconsistency detection and resolution for context-aware middleware support. In: Proceedings of the 10th European Software Sngineering Conference Held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 336–345 (2005)